Rapport de stage Master 2 : Apprentissage et étude de Transformer pour la classification en familles protéiques

Nicolas Buton

Rapport de stage Master 2 DAC
Apprentissage et étude de *Transformer* pour la classification en familles protéiques

Buton Nicolas
Sorbonne Université

Encadrants : François Coste, Yann Le Cunff
DYLISS/IRISA/Inria

Tuteur : Vincent Guigue
Sorbonne Université

January 8, 2021

Table des matières

Remerciements

Tout d'abord, je tiens à remercier l'institut IRISA, M.Coste et M.Le Cunff qui m'ont permis de faire ce stage, et qui ont pu m'aiguiller tout au long de celui-ci. Enfin, je souhaiterais remercier Marie Le Roïc qui m'a accompagné dans les démarches administratives.

Introduction

Les progrès en séquençage ont permis de connaître de plus en plus de séquences d'ADN, permettant de déduire les séquences des protéines qui sont produites par cet ADN. Il serait utile de connaître la fonction de ces protéines, qui est en grande partie déterminée par la structure 3D. Par exemple ces informations pourraient être utilisées pour comprendre comment perturber la "liaison" entre la spike protéine S1 du SARS-Cov-2 et le récepteur ACE2 des cellules humaines [11]. Malheureusement les expériences pour déduire la structure 3D et la fonction des protéines sont très coûteuses et chronophages. De fait, cela a créé une grande différence entre le nombre de protéines où l'on dispose de la séquence unidimensionnelle et le sous-ensemble où l'on possède aussi la structure 3D, c'est ce qu'on appelle le *sequence-structure gap*. Mais la structure 3D en elle-même ne nous intéresse pas trop, elle fait juste partie du pipeline biologique classique. C'est pour cela que l'on aimerait bien pouvoir déduire directement de la séquence protéique la fonction de celle-ci (fig 1.a), sans passer par la structure.

Certains modèles existent pour cette tâche mais ont d'importantes limitations. Par exemple nous avons Blast qui en alignant les séquences permet de calculer leurs degrés de similarité et donc nous pouvons ensuite utiliser des algorithmes comme le KNN pour classifier en familles fonctionnelles[1]. Nous avons aussi les profile Hidden Markov Model (pHMM), qui sont des modèles de Markov auxquels on a rajouté des états pour l'insertion et la délétion d'acides aminés. Ils permettent de caractériser une famille fonctionnelle mais sont malheureusement très limités pour modéliser les relations longue distance.

Dans le traitement automatique de la langue(TAL), il a été découvert une architecture de réseau de neurones nommée les Transformers qui a fait ses preuves pour les relations longue distance. Ces dites relations sont importantes pour notre problème, c'est pour cela que nous avons utilisé une architecture de Transformer. Ce qu'on entend ici par architecture, c'est la façon dont est connecté chaque neurone artificiel entre les différentes couches. Par exemple pour une architecture complètement connectée nous allons avoir des connexions entre tous les neurones de chaque couche. Ici l'architecture Transformer fonctionne sur le principe de l'attention, c'est-à-dire que pour construire la représentation d'un token on combine tous les tokens[2] auxquels il fait attention.

De plus nous disposons d'une grande quantité de séquences de protéines non annotées qui pourront être valorisées dans la première phase d'entraînement du Transformer. Cette architecture, fonctionnant grâce au mécanisme d'attention, compare tous les acides aminés entre eux ce qui permet de bien modéliser les relations longue distance entre ceux-ci. Enfin, il est possible de gérer l'expressivité du modèle assez simplement en modifiant ces hyperparamètres.

[1] Famille de protéines qui possèdent la même fonction
[2] Une ou plusieurs lettres regroupés

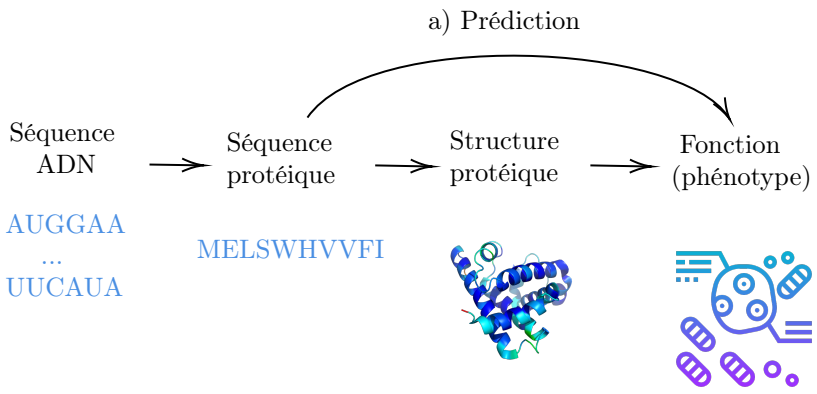


Figure 1: Schéma du pipeline biologique et informatique(a) classique passant de la séquence à la fonction d'une protéine

Dans ce stage, il a été validé que l'utilisation des Transformers était une bonne idée pour les protéines grâce à de nombreuses analyses du réseau post entraînement. Notamment nous avons remarqué que le plongement lexical fixe au départ du réseau nous permettait de rassembler les acides aminés ayant des propriétés physico-chimiques similaires. De plus, nous avons aussi remarqué que certaines têtes d'attention du réseau étaient focalisées sur les acides aminés étant proches dans la structure 3D, ce qui valide le fait qu'avec uniquement les données de séquence et avec une fonction de coût auto-encodeur l'on peut retrouver au moins en partie la conformation 3D de la protéine. Un autre avantage de l'architecture Transformer est que contrairement à beaucoup d'autres architectures en apprentissage profond, le réseau après entraînement n'est pas surconfiant avec des probabilités exagérément hautes et il représente très bien son incertitude. Différents tests ont permis de valider que le réseau est parfaitement calibré après entraînement.

En ce qui concerne la classification en familles protéiques on peut remarquer que l'on fait mieux que le précédent état de l'art avec des réseaux de neurones [3] qui utilisait un réseau convolutionnel [1] en passant de 72,3% d'accuracy à 78,8%. Ce résultat a été obtenu en utilisant un réseau Transformer pré entraîné ainsi qu'une projection vers le nombre de familles de protéines avec un dropout sur la représentation du token de classification. De plus il a été proposé de nouvelles architectures plus interprétables pour la localisation des acides aminés qui sont importants pour une classification en familles de protéines, au prix toutefois d'une baisse de performance de 3 points d'accuracy sur la base de donnée Pfam. Pour certains jeux de données nous avons aussi une structure d'ontologie sur les différentes classes fonctionnelles, par exemple avec la Gene Ontology(GO); mais malgré une modélisation avec des probabilités indépendantes on remarque que le réseau arrive très bien à comprendre la structure de l'ontologie et donc lorsque le modèle prédit un noeud il prédit bien aussi tous les parents de ce noeud[4].

En enregistrant quelques métriques au fur et à mesure de l'apprentissage nous avons remarqué que l'attention se spécialise au fur et à mesure. À l'initialisation du réseau Transformer, nous avons une attention très répartie entre tous les tokens mais ensuite l'attention se concentre vraiment sur très peu de tokens. De plus la matrice d'attention devient de moins en moins symétrique au fur et à mesure de l'apprentissage. C'est-à-dire que si le token A fait attention au token B, cela ne veut pas dire que le token B fait attention au token A. De plus on a pu remarquer que lors de l'apprentissage le Transformer commençait par se focaliser sur la prédiction de l'acide aminé le plus commun et ensuite il prédisait le second plus fréquent et ainsi de suite[5], mais en général même après beaucoup d'apprentissage il sur-prédit les acides aminés les plus fréquents et sous-prédit les acides aminés les moins fréquents.

Ces outils informatiques pour déterminer la fonction d'une séquence protéique peuvent être utiles dans beaucoup de domaines de la biologie. Cela peut aider pour avoir les premières pistes de protéines à étudier plus en détails dans le cadre d'une maladie ou d'un autre phénomène que l'on veut comprendre. Comme il

[3]Sans méthode d'ensemble
[4]On retrouve environ 99% des noeuds parents avec un seuil de probabilité de 0.5
[5]Pas toujours exactement dans l'ordre des fréquences mais très proche

est impossible de réaliser des expériences sur toutes les protéines produites par un organisme, il est important de pré-sélectionner un nombre réduit de celles-ci pour une analyse plus approfondie.

Ce rapport est découpé en 6 parties. Dans un premier temps je vais présenter le laboratoire de recherche, l'organisation de celui-ci ainsi qu'une présentation du département et de l'équipe dont je faisais partie. Ensuite, je présenterai l'état de l'art en Traitement Automatique des Langues(TAL) et en caractérisation de protéines. Ces dernières années nous avons eu beaucoup d'améliorations dans les performances sur plusieurs tâches de TAL, notamment grâce à l'architecture des Transformers. Dans un troisième temps nous verrons quelles librairies, matériel et architectures de réseaux ont été utilisés, pour ensuite s'attarder sur les résultats obtenus lors de ce stage. Enfin on en évoquera les perspectives possibles.

Définitions d'outils

Ci-dessous nous allons définir les outils mathématiques utilisés dans la suite de ce rapport :

1: Divergence de Kullback-Leibler

Permet de calculer une mesure de similarité entre deux distributions de probabilité.

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

2: Coût cross entropique

Pour une seule entrée :

$$CE = -\sum_{c=1}^{M} y_c \log P_c$$

y_c : 1 si la classe c est correcte sinon 0
P_c : Probabilité de prédiction de la classe c

3: Similarité cosinus

Pour comparer nos vecteurs, nous utiliserons ces différents outils dans la partie méthode :

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

1 Présentation du laboratoire de recherche

J'ai eu l'occasion de réaliser mon stage au sein de l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires). Cet institut regroupe huit établissements de recherche : Centrale Supélec, CNRS, ENS Rennes, IMT Atlantique, Inria, INSA Rennes, Université de Bretagne Sud (UBS), Université de Rennes 1 . Il y a 800 personnes qui y travaillent et celles-ci sont réparties en 40 équipes et sept départements (Systèmes large échelle / réseau, télécommunication et services / Architecture / Langage et génie logiciel /Signaux et images numériques, robotique / Média et interactions / Gestion des données et de la connaissance). Il y a aussi des axes transversaux qui ont une implication dans plusieurs départements(Cyber-sécurité, Biologie Santé, Environnement Ecologie, Green IT , Patrimoine & Culture, Robotique & Drone, Transport). Ce sont des sujets plus globaux, de grands sujets sociétaux , ils permettent de favoriser les collaborations transversales. Toutes les équipes ont une durée de vie limitée qui est définie dès leur création. Cela permet de renouveler les idées, de partir sur de nouvelles bases et de changer l'organisation. L'IRISA possède plusieurs partenaires industriels français : EADS, Orange, EDF, Nexter System, Renault, ST Microelectronics, Thales, Technicolor, Alcatel, mais aussi des partenaires étrangers : exasInstr., IBM, Google, Intel. Elle permet également la création de nouvelles startups telles que : KERLABS, EVODIA, GOALEM, CAPS entreprise, SenseYou, Mensia, SYNEIKA.

Je faisais partie du département Gestion des données et de la connaissance qui s'intéresse au traitement des données et se donne plus précisément pour objectif d'explorer les relations entre données et connaissances. Les travaux portent sur le stockage, l'interrogation et la visualisation de données massives ou complexes, ainsi que sur l'exploitation et la valorisation de ces données. J'étais plus particulièrement dans l'équipe de recherche DYLISS dont le but de recherche est de modéliser les acteurs clés de processus d'adaptation biologique, en utilisant des systèmes formels pour structurer et combiner les informations issues de données génomiques et physiologiques.

2 État de l'art

Dans cette partie nous allons commencer par décrire la tâche de classification en familles de protéines, ensuite on verra les dernières avancées en TAL puis leurs applications pour la caractérisation de protéines.

2.1 Classification en familles de protéines

L'Acide désoxyribonucléique (ADN) est une molécule qui sert de support à l'information biologique. Cette molécule est un assemblage de 4 bases azotées : l'Adénine, la Cytosine, la Guanine, la Thymine. À partir de ces bases nous pouvons en déduire la séquence des protéines grâce au code génétique, qui à chaque triplet de nucléotides associe un acide aminé. Les protéines sont des molécules composées d'acides aminés (il y en a 20 classiques) qui sont présentes dans toutes les cellules vivantes, et sont impliquées dans la majorité des fonctions que la cellule effectue. C'est pour cela que l'on aimerait les classer en familles qui possèdent la même fonction.

Parmi les outils classiques utilisés en bio-informatique pour cette tâche, il y a notamment Blast et les pHMM. Blast permet d'aligner deux séquences pour ensuite en déduire un degré de ressemblance. L'alignement consiste à introduire des trous à certains endroits pour pouvoir avoir le maximum de lettres en commun à la même position (exemple Figure 2). Ceci peut être utilisé pour caractériser de nouvelle protéines en lui associant la fonction de la protéine (annotée expérimentalement) avec le plus grand degré de ressemblance. Les pHMM, eux, fonctionnent différemment. Ils sont capables de caractériser une famille fonctionnelle et d'associer une probabilité d'appartenance à cette famille. Pour caractériser une nouvelle protéine on peut calculer la probabilité d'appartenance à chaque famille fonctionnelle définie par des pHMM.

Mais ces modèles classique ont des limites. Blast ne caractérise pas directement la fonction et les pHMM font une hypothèse de Markov d'ordre 1[6], ce qui ne permet de pas de bien modéliser les relations longue

[6]La probabilité de l'acide aminé N dépend uniquement de l'acide aminé à la position N-1

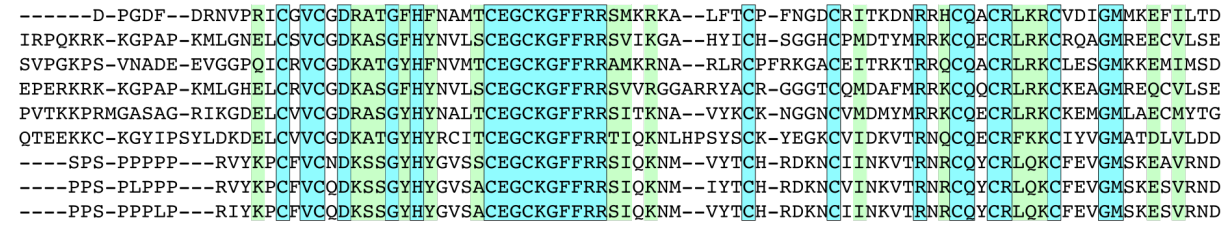


Figure 2: Exemple d'alignement multiple de séquences protéiques

distance. Or ces relations sont importantes car dans la structure 3D nous pouvons avoir deux acides aminées très éloignés dans la séquence mais qui sont en contact direct, ce qui va donc impliquer une conformation spécifique qui influencera fortement la fonction. C'est pour cela que nous avons besoin de nouveaux modèles pour prédire la fonction d'une protéine uniquement à partir de sa séquence. Et donc nous allons voir les dernières avancées en TAL qui pourront être utilisées dans ce cadre.

2.2 Présentation des jeux de données

La première base de données utilisée a été la base Pfam, qui classifie les domaines des protéines en familles. Un domaine est une sous-partie fonctionnelle de la séquence protéique. Cette base de données a été construite de façon itérative, c'est-à-dire que premièrement des biologistes sélectionnent un ensemble de domaines qu'ils considèrent appartenir à la même famille, ensuite ils construisent un pHMM à partir de ces séquences. Avec ce pHMM ils testent toutes les protéines et récupèrent uniquement celles avec les plus grandes probabilités d'appartenance. Ils sélectionnent ensuite celles qui appartiennent toujours à la même famille pour construire un nouveau pHMM. Et ainsi de suite jusqu'à arriver à un modèle stable.[7] Dans cette base de données il y a deux parties. Une première où l'on a les protéines vérifiées par les biologistes qui est *pfam-seed* et la seconde partie qui est juste le résultat des pHMM créés sur beaucoup plus de protéines mais non vérifiées. Par la suite nous allons principalement nous intéresser à pfam-seed qui contient les protéines vérifiées expérimentalement.

Pour la phase de pré entraînement, nous avons besoin de beaucoup de séquences de protéines. Nous avons donc 3 jeux de données distincts. Le premier contient les séquences issues des domaines pfam, le second est composé des séquences de toutes les protéines présentes dans pfam-seed et le dernier est Uniparc qui est une plus grosse base de données de séquences de protéines avec 330 millions de séquences. La longueur moyenne des domaines est d'environ 150 acides aminés et la longueur moyenne des protéines est de 350 acides aminés.

À partir de la base de données pfam-seed j'ai aussi eu l'occasion de créer un nouveau jeu de données pour la localisation de domaines(ref figure 3). C'est-à-dire que l'on associe à une protéine complète le début, la fin et l'appartenance à la famille de domaines pour chacun de ceux-ci.

Figure 3: Séquence protéique annotée avec les domaines Pfam

Un benchmark nommé TAPE[18] a été proposé pour regrouper diverses tâches sur les protéines. Il y a de la prédiction de structure secondaire[8] où chaque acide aminé doit être classé dans une de ces 3 catégories : Helix, Strand, Other; de la prédiction de classe de structure où ici l'on doit associer une seule classe pour toute la protéine. Mais aussi de la prédiction de contact pour savoir si deux acides aminés sont

[7] Ceci introduit un biais dans la construction car l'on retrouve plus facilement les protéines qui sont facilement identifiables avec des pHMM pour une famille donnée.

[8] Structure locale intermédiaire avant la structure 3D

à moins de 8 Å l'un de l'autre. Puis des tâches plus spécifiques avec de la prédiction de fluorescence de la protéine ou la stabilité de celle-ci, qui sont des tâches de régression[9]. Il est difficile de faire un modèle qui a de bonnes performances sur cette diversité de tâches, c'est en cela que ce jeu de données est intéressant.

D'autres jeux de données sont spécifiquement orientés pour la prédiction de fonction, par exemple le jeu de données de CAFA[10]. On dispose de 3 ontologies, pour 3 façons de classer les protéines. Soit par fonctions moléculaires(MFO) qui sont très bien définies, par fonctions biologiques(BPO) qui sont plus larges et souvent un peu plus floues, et par emplacement dans la structure cellulaire(CCO).Le challenge CAFA, qui a déjà plusieurs éditions à son actif, demande à ses participants de produire leur algorithme de prédiction à partir de toutes les données qu'ils jugent utiles. Pendant le concours, un certain nombre de prédictions sont produites sur des protéines non testées expérimentalement. Plusieurs mois plus tard, ces protéines sont testées expérimentalement ce qui permet de comparer les données prédictives et expérimentales. Avec cette méthode, on est sûr que le jeu de test n'a pas fuité puisque personne ne connaît la réponse au moment des prédictions des modèles.

Enfin pour les informations de structure 3D des protéines nous avons la base de données PDB(Protein Data Bank), où l'on peut retrouver beaucoup de structures déterminées expérimentalement.

2.3 Données biologiques a priori

Plusieurs outils ont été développés précédemment pour travailler sur les séquences biologiques. Nous allons en définir quelque-uns qui nous serviront ensuite dans l'analyse de nos réseaux de neurones entraînés. Nous avons par exemple les position-specific scoring matrix(PSSM), qui sont des matrices de taille nombre de symboles possibles $\times$ taille de la séquence, et où la somme de chaque colonne est 1. Cela représente la probabilité d'apparition de chaque symbole à chaque position. On peut rajouter un a priori à combiner avec la fréquence des symboles si notre jeu de données est trop petit. Comme outils, nous avons aussi les matrices de substitution qui représentent, la plupart du temps sur des protéines qui sont liées évolutionnairement, en quoi se change chaque acide aminé. Nous avons par exemple la matrice de substitution Blosum62, qui prend tout les blocs conservés sans insertion avec plus de 62% d'identité, on peut changer le pourcentage d'identité pour avoir d'autres matrices Blosum mais la plupart du temps on garde 62% qui est un bon compromis entre dépendance locale et distante. HMMER est un programme qui utilise notamment les pHMM pour retrouver des séquences homologues [11].

2.4 Dernières avancées en apprentissage automatique sur le TAL

Dernièrement nous avons vu une architecture qui fonctionne très bien pour les relations longue distance, qui est l'architecture Transformer. Elle permet de traiter des relations entre des mots éloignés mieux que des réseaux récurrents [4]. En complément de la nouvelle architecture de réseaux, il y a eu le développement de réseaux avec deux phases d'apprentissage [8]. La première, le pré entraînement, très coûteux mais à faire une seule fois, permet de créer un réseau qui a une certaine compréhension générale de la langue. Ensuite la seconde phase que l'on nomme phase d'ajustement (*fine-tuning*) permet d'entraîner le réseau sur une tâche spécifique (par exemple de la classification de documents / ressemblance sémantique / analyse de sentiment)[23]. En plus de détailler ce qui se fait dans ces deux phases nous verrons aussi les différentes variantes de l'architecture Transformer.

2.4.1 Pré entraînement

Fonction de coût d'apprentissage

[9] Il faut prédire un chiffre réel

[10] The Critical Assessment of protein Function Annotation algorithms

[11] Des séquence homologues sont des séquences qui partagent une origine évolutive commune

Le premier modèle à utiliser avec succès l'architecture Transformer [16] utilisait uniquement la partie décodeur de l'architecture. Ce modèle a été conçu pour modéliser la probabilité du prochain mot sachant les mots précédents. C'est donc une fonction de coût auto-régressif, qui ne prend donc en compte le contexte que d'un côté. Par la suite nous avons eu des modèles qui permettent un encodage bidirectionnel [7] avec une fonction de coût auto encodeur, c'est-à-dire qu'à partir d'une entrée corrompue (On remplace certains tokens[12] par le token <mask>), on essaye de reconstruire la vraie entrée. En complément de cette fonction de coût, une tâche de prédiction de la prochaine phrase à été introduite. C'est une fonction de coût de classification binaire pour savoir si deux phrases se suivent. Parfois on prend deux phrases réellement consécutives et parfois on tire deux phrases au hasard dans le corpus. Cette fonction de coût permet d'avoir une compréhension plus haut niveau, mais ne fait pas consensus dans le domaine car il a été montré que cette fonction de coût dégradait les performances sur certaines tâches [25]. Plusieurs variantes de cette fonction de coût ont été proposées, par exemple où l'on inverse juste les deux phrases [12], ce qui évite au réseau de prédire uniquement le contexte et de remarquer un changement drastique de celui-ci.

La fonction de coût avec le masque a aussi été critiquée car le token <mask> n'apparaîtra jamais dans les données et de plus le coût auto-encodeur suppose l'indépendance des termes masqués, ce qui n'est pas satisfaisant. C'est pour cela qu'il a été proposé une nouvelle fonction de coût auto régressif généralisé avec un contexte bidirectionnel [25]. Celui-ci fonctionne en permutant aléatoirement les entrées et en essayant de prédire le mot avec tous les mots précédents dans la permutation(voir fig **??**), en sachant que t commence a partir d'un certain c, qui est un hyperparamètre qui définit combien de mots de contexte sont nécessaires pour prédire ceux qui nous manquent. L'ordre permuté est utilisé uniquement pour savoir quel mot est visible pour prédire les autres mots mais ne change en aucun cas l'ordre d'entrée dans le Transformer.

Il y a aussi d'autres variantes de fonction de coût. Par exemple une fonction de coût de *text-infilling* [13] qui consiste à tirer une longueur selon une loi de Poisson($\lambda = 3$) puis tirer aléatoirement des emplacements et retirer la longueur tirée de token pour la représenter par un masque. Avec cette fonction de coût, le réseau doit retrouver les tokens manquants mais aussi retrouver combien de tokens il manque. Une autre fonction de coût [24], où l'on remplace 15% des tokens par le token <mask> puis on sélectionne 5% des tri-grammes en permutant aléatoirement l'ordre et on doit re-prédire les tokens dans l'ordre. Aussi une nouvelle fonction de coût pour l'ordre des phrases avec une classification en 3 groupes a été proposée; groupe 1 : S1 et S2 sont dans le bon ordre, groupe 2 : S1 et S2 sont dans l'ordre inversé, ou groupe 3 : S2 a été échantillonné aléatoirement dans le corpus.

Certaines méthodes encore peu testées essayent de rendre l'entraînement plus efficace en temps de calcul en proposant une nouvelle manière d'entraîner les Transformers[5] en introduisant la fonction de coût de détection de token. L'entraînement nécessite deux réseaux. Un premier, le générateur, qui à partir d'entrées masquées va prédire une phrase en prédisant les mots masqués et un deuxième réseau, le discriminateur (celui que l'on gardera après la phase de pré entraînement), qui doit discriminer si l'entrée est réelle ou a été générée par le réseau générateur. Cela permet au réseau discriminateur d'être entraîné avec vraiment toutes les prédictions et pas uniquement à l'emplacement des <mask> tokens, ce qui rend l'entraînement plus efficace. Le générateur possède beaucoup moins de paramètres que le discriminateur et donc n'ajoute qu'un faible surcoût à l'apprentissage.

Étude des hyperparamètres

Il a été montré que même avec une puissance de calcul importante et un temps d'entraînement long, la plupart des modèles sont encore en sous apprentissage à la fin de leur entraînement [15]. C'est pour cela qu'il faut entraîner ces réseaux Transformer très longtemps.

Quelques résultats prometteurs montrent [17], [13] qu'il serait mieux d'utiliser toute l'architecture du Transformer plutôt que de se limiter uniquement à la partie encodeur ou décodeur. En effet, on peut entraîner le réseau sans rajouter de couches linéaires pour chaque tâche différente et traiter toutes les tâches

[12] Un token est composé d'une ou plusieurs lettres

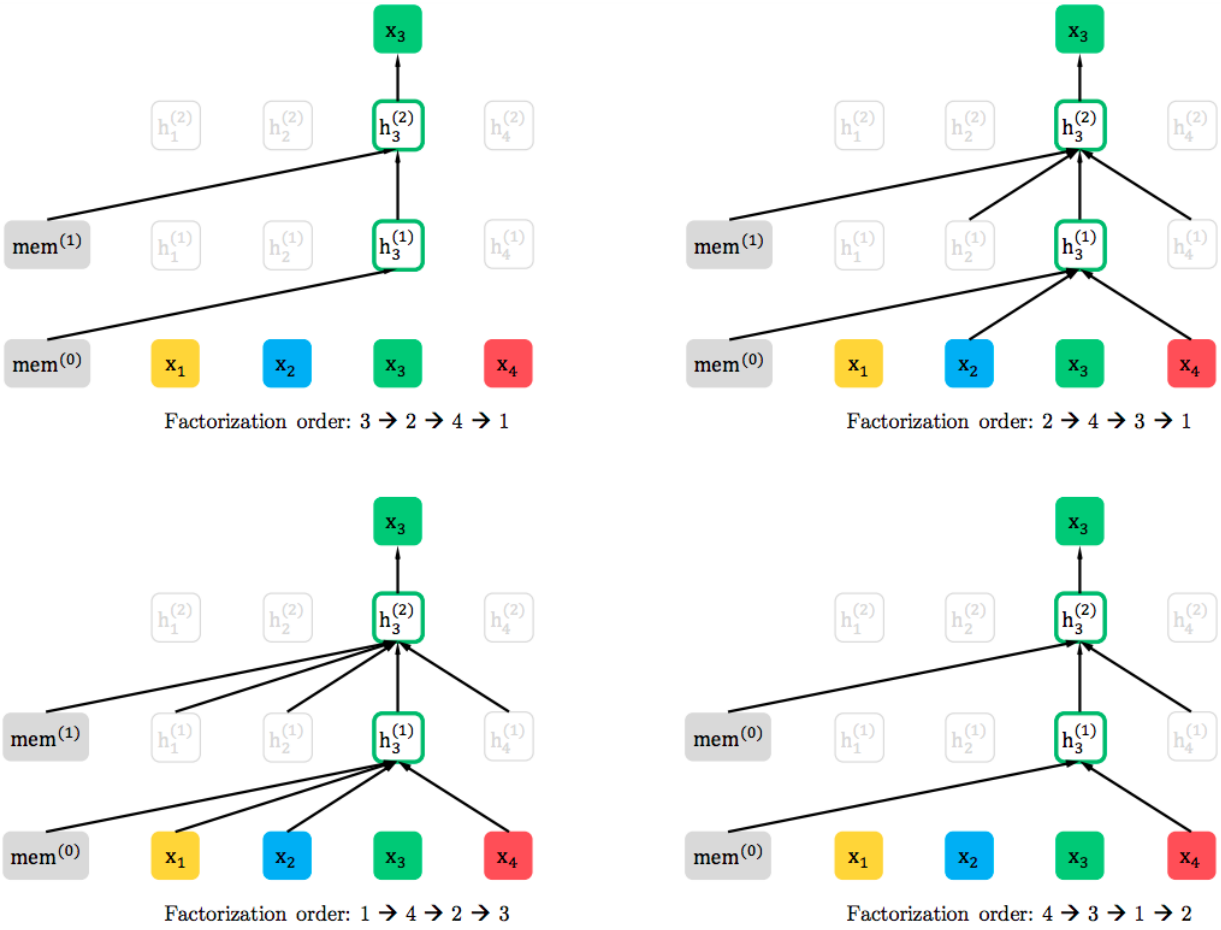


Figure 4: Fonction de coût auto régressive généralisé présenté avec l'architecture XLNET

comme une prédiction texte vers texte. Lorsque c'est de la régression, on peut discrétiser l'ensemble. Des tests[17] sur une grande variété d'hyperparamètres ont montré que ce qui est le plus significatif est le nombre de paramètres du réseau, la quantité de calcul dépensée et la taille des jeux de données. Mais même s'ils sont moins conséquent en terme d'amélioration, il y a d'autre hyperparamètres qui permettent d'avoir de meilleures performances. Il se trouve que pré entraîner avec une fonction de coût non supervisée sur des données issues de la même distribution que celle d'une tâche particulière améliore les résultats. De plus, le *span-corruption objective* serait un bon compromis entre performance et temps de calcul. Cet objectif consiste à sélectionner aléatoirement des ensembles de l mots d'affilée et de les remplacer par un unique masque[13]).

2.4.2 Phase d'ajustement

Pour la phase d'ajustement certaines techniques ont été développées pour améliorer les résultats, notamment l'utilisation des ensembles de réseaux [14]. À partir d'un réseau pré entraîné, on initialise N réseaux pour chaque tâche qu'on entraîne. Ensuite on crée un autre réseau multi-tâches qui a une fonction de coût en deux parties. La première essaye de se rapprocher de la distribution moyenne des N réseaux déjà entraînés pour chaque tâche, et la seconde partie essaye de se rapprocher de la vraie classe. C'est de la distillation de connaissance des N réseaux vers l'unique réseau final.

Il a aussi été proposé d'entraîner le même réseau sur plusieurs tâches en même temps [22], avec plusieurs tâches pour la phase de pré apprentissage. Ces tâches sont réparties en 3 grands axes "Word-aware", "Structure-aware" et "Semantic-aware". Ils pré entraînent le réseau d'abord sur la tâche 1, ensuite sur la tâche 1 et 2, ensuite 1,2 et 3 et ainsi de suite. Chaque tâche sera entraînée au total uniquement sur N itérations donc il faut répartir les itérations. Cela permet de ne pas oublier les anciennes tâches.

2.4.3 Variante de l'architecture

Plusieurs variantes de l'architecture des Transformers ont été proposées. Par exemple, les Transformers n'étant pas récursifs, contrairement aux RNNs, ils ont une entrée de taille fixe. C'est pour cela qu'il a été proposé d'utiliser un mécanisme de récursion dans les Transformers qui permet d'utiliser des séquences de taille quelconque en entrée [6]. Ce mécanisme de récursion fonctionne en gardant les états cachés de l'exécution du Transformer sur le segment précédent(segment d'environ 300 mots), et concatène l'état caché de l'exécution en cours et les anciens états cachés, puis crée les vecteurs clés, requêtes et valeurs à partir de cette concaténation. Cela a permis dans le même temps l'introduction d'un positionnement relatif intégré au réseau car un positionnement global comme avant n'était plus possible.

Au vu des temps nécessaires en apprentissage mais aussi en inférence car les réseaux ont parfois des milliards de paramètres (175 milliards pour certains [2]), il a été proposé des réseaux plus légers qui gardent presque la même performance que les modèles sur lequel ils sont basés [20], [9]. Ces réseaux ont un nombre de couches réduit, sont initialisés avec une partie des poids du réseau original et sont entraînés avec la fonction de coût auto encodeur mais aussi avec une fonction de coût qui essaye de rapprocher la sortie d'un réseaux original[14]. Avec cette technique, certains[20] ont réussi à conserver 96% des performances tout en réduisant le nombre de paramètres de 28%. Mais ces modèles nécessitaient toujours un réseau original complet et pré entraîné longtemps. C'est pour cela que certains ont décidé de se passer du réseau original [12] en partageant tous les paramètres de toutes les couches du Transformer et en réduisant le nombre de paramètres pour le plongement lexical en les factorisant[15]. Cela permet d'avoir 3,9 millions de paramètres au lieu de 23 millions dans cette partie.

[13] l égal à 3 serait un bon choix d'après [17]

[14] On fait la Divergence Kullback-Leibler entre la sortie du réseaux originale et la sortie du petit réseau. Cette distribution complète est utile car elle apporte plus d'information qu'avec uniquement le token sur cet exemple.

[15] Par exemple pour BERT au lieu de passer directement de la dimension du vocabulaire(30 000 par exemple) à la dimension du plongement lexical (768 par exemple), on passe à une dimension intermédiaire de taille 128

Avec les Transformers, nous ne pouvons pas traiter des séquences trop grandes car l'attention compare tous les tokens entre eux ce qui fait une complexité en $O(n^2)$. Certaines variantes des Transformers ont été proposées [10], notamment en partageant les clés et les requêtes ou en utilisant une attention LSH[16] qui ne calcule l'attention qu'entre les tokens qui se ressemblent assez. Ils découvrent ceux qui se ressemblent assez en un temps raisonnable grâce a un hashage qui garde un hash proche pour les entrée proche. Cela réduit la complexité de calcul de l'attention a $O(n * logn)$. Nous avons aussi le papier Sparse Transformer [3], qui réduit la complexité de l'attention en $O(n * \sqrt[p]{n})$. Pour ce faire, les chercheurs calculent avec des têtes d'attention locale se concentrant uniquement sur les tokens autour avec une distance de l et une autre tête où tous les l tokens font attention à tous les autres tokens. Ceci permet de propager l'attention si on a besoin d'un contexte plus grand que l.

L'impact mémoire est aussi très important lorsque l'on fait tourner un Transformer, c'est pour cela qu'il a été proposé [10] quelques astuces là-dessus. Par exemple, des couches réversibles qui permettent de sauvegarder uniquement le dernier état caché pour réussir à faire le backward au lieu de tous les sauvegarder. Ils effectuent aussi un découpage en morceaux pour le feed-forward car ils sont calculés indépendemment.

Nous allons maintenant voir comment on peut appliquer tout cela à la caractérisation de protéines.

2.5 Premières applications de l'apprentissage profond sur la caractérisation de protéines

2.5.1 Création des jeux de données pour les protéines

Dans le domaines des protéines nous avons beaucoup de similarité entre séquences. Pour tester les capacités de généralisation du réseau, nous ne pouvons donc pas faire des séparations aléatoires en ensembles de test, développement et apprentissage comme en TAL. Sinon nous allons avoir des séquences avec très peu d'acides aminés qui changent mais certaines seront dans l'ensemble de test et d'autres dans l'ensemble d'entraînement. Pour améliorer ce point, il faut d'abord faire un partitionnement de données sur les séquences selon un pourcentage de similarité et ensuite sélectionner aléatoirement les clusters pour les associer aux différents ensembles. Par exemple, pour une tâche de classification sur Pfam[1], on peut passer de 0.15% d'erreur sur une séparation aléatoire à 12% avec une séparation avec partitionnement de données.

2.5.2 Fuite de données entre pré entraînement et phase d'ajustement

Dans la phase de pré entraînement, on utilise les séquences de toutes les protéines et potentiellement il y a des séquences qui seront dans ce jeu de données de pré entraînement sans label mais qui seront aussi présentes dans le jeu de données de test pour la phase d'ajustement. Pour tester l'impact de ceci, certains [21] ont pré entraîné en séparant par clusters de similarité séquentielle ou en faisant un tirage uniforme sur les séquences. Et on peut remarquer que les métriques de pré entraînement (ECE/Perplexity/Accuracy) sont meilleures avec l'uniforme mais cela ne donne pas de meilleures performances sur les tâches de classification. Donc on peut en déduire que la fuite de données à un faible impact sur les tâches qui nous intéressent vraiment: les phases d'ajustements.

2.5.3 Architecture testée

La classification en famille pfam a été testée avec un CNN [1]. Le réseau est composé d'une première partie qui permet d'avoir un plongement lexical des protéines dans un espace de dimension N et ensuite d'une partie de réseaux pour la classification. Il a aussi été testé de retirer la partie pour la classification et de faire deux variantes. La première en prenant un vecteur moyen par famille et pour chaque séquence du test on regarde le vecteur moyen le plus proche en terme de distance cosinus(Per-Familly). Un deuxième modèle où au lieu de construire les vecteurs moyens on construit un vecteur pour chaque élément du test et on prend le label du plus proche(Per-Instance). Le per-familly permet de meilleurs résultats lorsque les

[16] Locality sensitive hashing

familles comportent peu d'exemples, mais pour les plus grandes familles le Per-Instance est meilleur. Avec la méthode de per-familly on peut tester de prédire la classification sur de nouvelles familles en donnant comme vecteur moyen une ou plusieurs nouvelles séquences pour la représenter. À partir de deux séquences pour une famille il est possible d'avoir de meilleurs résultats que la baseline HMMER.

En explorant ce CNN on retrouve des informations cohérentes avec les connaissances biologiques, par exemple la matrice de plongement lexical ressemble à la matrice de substition BLOSMU62. Et si l'on essaye de remplacer dans une protéine bien connue "ATPase" certains acides aminés par d'autres et que l'on fait la divergence de Kullback-Leibler entre les distributions sur les classes, d'une part la distribution avec la séquence originale et d'autre part la séquence avec la modification d'un acide aminé, on remarque une grande différence si l'on touche aux sites qui participent à la fonction. Certaine substitution sont également meilleures que d'autres si l'acide aminé a les bonnes propriétés, ce qui est cohérent avec les données biologiques.

Pour éviter de sur-estimer les capacités de généralisation, il est possible de tracer les courbes de taux d'erreur en fonction du degré de ressemblance avec l'ensemble d'entraînement [1].

L'architecture Transformer a aussi été testée récemment en l'entraînant sur 250 millions de séquences protéiques [19]. Les auteurs remarquent que l'espace résultant est organisé en plusieurs niveaux de granularité. Par exemple, en projetant la matrice qui sert pour le décodage dans un espace à deux dimensions avec t-SNE, on remarque une séparation selon les propriétés physico-chimiques (aromatique / hydrophile/ polaire). Cependant, nous avons aussi des informations au niveau de la protéine. Avec une projection linéaire, on peut retrouver des informations sur la structure secondaire et les contacts entre les différents acides aminés.

2.5.4 Biais inductif d'un Transformer

On remarque que même un Transformer non entraîné sépare beaucoup mieux les familles homologues que le LSTM[17]. Les LSTM sont des réseaux de neurones récurrents qui étaient beaucoup utilisés en apprentissage profond sur les séquences avant l'apparition des Transformers et sont encore maintenant utilisés pour certaines applications.

On peut aussi tester des opérations sur les vecteurs générés par le Transformer. Par exemple on peut extraire un vecteur moyen qui passe de la protéine A à la protéine B, peu importe l'espèce vivante. On teste ensuite ce vecteur moyen sur d'autres espèces[18] en ajoutant le vecteur à A et en retrouvant les K plus proches voisins de ces nouvelles coordonnées pour essayer de retrouver B. On peut faire quelque chose de similaire aussi pour passer d'une espèce e1 à une espèce e2, peu importe la protéine cette fois-ci. Tout ceci fonctionne plutôt bien avec un Transformer entraîné.

2.5.5 Rassemblement des acides aminés pour former des tokens

Il a été testé[21] d'utiliser BPE(*byte-pair-encoding*) sur les acides aminés des protéines mais on ne peut pas directement comparer les métriques de pré entraînement car elles dépendent de la taille du vocabulaire. Cependant, en regardant les performances sur des tâches spécifiques on peut remarquer que ça ne fonctionne pas mieux.

[17] En projetant dans un espace à 2 dimensions avec t-SNE et PCA et en colorant les gènes homologues avec les deux architectures, on peut remarquer une séparation plus distincte avec le Transformer.

[18] exclues de l'ensemble qui à été choisi pour calculer le vecteur moyen

3 Matériel et méthode

Dans cette partie nous allons voir les outils utilisés pour mener à bien ce stage, ainsi que les architectures proposées, les méthodes d'entraînement et la façon d'évaluer les différents modèles.

3.1 Matériel et logiciels/librairies

Tout d'abord pour avoir un historique du code j'ai utilisé le gitlab de l'Inria. J'ai utilisé Python avec la librairie pytorch pour développer et travailler sur les réseaux de neurones. Dans le projet gitlab j'ai créé un environnement python pour pouvoir facilement activer cet environnement et exécuter le code. De plus ayant eu besoin de beaucoup de puissance de calcul, j'ai pu accéder à des GPU Nvidia V100 sur un cluster géré par l'équipe GenOuest qui utilise SLURM comme outils pour gérer les listes d'attente pour l'accès au matériel.

3.2 Évaluation du pré entraînement

La phase de pré entraînement s'est faite avec une fonction de coût cross entropique où l'on essayait de prédire 15% des tokens. Les tokens étaient masqués dans 80% des cas, remplacés par un token aléatoire dans 10% des cas et laissés tels quels le reste du temps. Pour commencer l'évaluation du réseau pré entraîné, on visualise aussi les courbes de fonction de coût au fur et à mesure de l'apprentissage. Après, on a récupéré la matrice de plongement lexical des acides aminés, et l'on a calculé une matrice de similarité/distance selon plusieurs métriques : la similarité cosinus, la distance euclidienne, le produit scalaire. L'objectif est d'obtenir une grand similarité pour les acides aminés qui ont des propriétés physico-chimiques proches, par exemple la polarité, la masse, hydrophobie, la charge et l'aromatisation. On a aussi regardé si on retrouvait des clusters par propriété avec une projection à l'aide de t-SNE et UMAP[19].

Ensuite on regarde la distribution de l'entropie à l'intérieur et à l'extérieur des domaines. On espère voir une entropie plus importante hors des domaines car ce sont des régions moins conservées par l'évolution. De plus nous avons deux façons de faire, la première en regardant l'entropie de prédiction de tous les acides aminées sans rien masquer, ou sinon en masquant certain acide aminé et en ne regardant l'entropie que de ces acides aminés masqués. On peut aussi calculer la divergence de Kullback-Leibler avec la distribution qui représente l'a-priori, c'est-à-dire les fréquences d'apparition des acides aminés dans le jeu de données d'apprentissage.

On peut aussi regarder la distribution des prédictions par rapport aux fréquences d'apparition des acides aminés pour voir si l'on se rapproche des fréquences ou si la distribution est encore très différente. Enfin on peut faire des tests de calibration pour voir si lorsque le modèle est sûr de lui alors la réponse a vraiment plus de chance d'être correcte, c'est-à-dire s'il estime bien les probabilités sur les différents acides aminés.

On s'attardera aussi à vérifier si l'on peut retrouver des informations sur la structure 3D après la phase de pré entraînement, par exemple, les informations de contact. On définit un contact entre deux acides aminés s'ils sont à moins de 8 Å. Ensuite nous les séparerons en 3 types pour les confronter : les contacts proches aussi dans la séquence, les contacts uniquement lointains dans la séquence et l'union des deux types précédents de contacts. Pour l'attention nous aurons aussi 3 façons de l'utiliser, la première en prenant un seuil sur l'attention maximale sur toutes les têtes, la seconde avec un seuil sur chaque tête puis une agrégation des matrices de confusion, et la dernière en prenant la tête ayant le meilleur f1 toujours avec un seuil sur chaque attention.

3.3 Mise en place du pipeline d'apprentissage

Pour entraîner les réseaux de neurones j'ai choisi de recoder le pipeline d'apprentissage et les réseaux plutôt que de prendre des librairies comme fairSeq ou HuggingFace. Cela m'a permis d'avoir une meilleur maîtrise de ce qui est fait, ainsi que d'avoir la possibilité de changer chaque détail, par exemple la façon dont le pré

[19] Algorithme de réduction de dimensions qui permettent par exemple la visualisation en 2D.

apprentissage est fait, la construction de la tâche non-supervisée, etc au prix toutefois d'une moins bonne optimisation et donc d'un code qui met un peu plus longtemps à s'exécuter. Cela m'a aussi permis de choisir ce que je voulais sauvegarder au cours de l'apprentissage et à quelle fréquence. J'ai utilisé la classe DataParallel de pytorch qui permet une parallélisation rapide sur plusieurs GPU.

3.4 Architectures pour la classification

Pour la partie classification après le pré entraînement il a été testé plusieurs variantes. Certaines en projetant directement la représentation du vecteur <CLS> vers le nombre de classes, d'autre en passant par un espace intermédiaire plus grand pour tester si une plus grande expressivité du modèle peut être utile pour la phase de classification. Ensuite dans d'autres variantes on projette chaque représentation de chaque acide aminé vers le nombre de classes pour ensuite faire un pooling afin d'avoir juste une décision sur le nombre de classes. On a testé un pooling max et moyen.

3.5 Localisation des domaines

J'ai choisi de commencer par une architecture de type Transformer avec uniquement l'encodeur, avec une fonction de coût autoencoder comme pour BERT, et un vocabulaire qui contient chaque type d'acides aminés, ainsi que quelques tokens spéciaux[20]. Pour la phase d'ajustement, je vais par la suite proposer 2 variantes de la fin de l'architecture, dont l'objectif est d'avoir un réseau plus interprétable, notamment au niveau de la localisation des acides aminés ce qui est important pour la classification dans les familles de protéines.

La première variante est classique, c'est-à-dire que l'on projette uniqueent le vecteur du token <CLS> vers le nombre de classes. Dans la seconde, à la place de réserver un seul token spécifique pour la classification comme le token <cls>, on classifie le plongement lexical de sortie de chaque acide aminé. Ensuite on effectue un pooling[21] afin de déduire la classe du domaine en entrée(Figure 5). Avec ces architectures nous présenterons 3 façons d'extraire des informations sur la localisation de la décision par domaine.

Attention par rapport au token CLS

On peut visualiser les différentes attentions sur la dernière couche par rapport au token <cls>, puis essayer d'agréger les têtes d'attentions avec une moyenne, médiane.

Visualisation de l'attention par classe grâce au gradient

On peut voir la probabilité finale d'une classe C_i comme résultant d'un fonction $f(t_1, .., t_L, \theta)$, avec θ les paramètres du réseaux,et t_i les différents tokens on peut ensuite calculer :

$$Df_i = \frac{\partial f(t_1, .., t_L, \theta)}{\partial t_i} = \begin{pmatrix} \frac{\partial f(t_1, .., t_L, \theta)}{\partial t_{i,1}} \\ \frac{\partial f(t_1, .., t_L, \theta)}{\partial t_{i,2}} \\ \vdots \\ \frac{\partial f(t_1, .., t_L, \theta)}{\partial t_{i,j}} \\ \vdots \\ \frac{\partial f(t_1, .., t_L, \theta)}{\partial t_{i,n}} \end{pmatrix} \tag{1}$$

[20] <CLS>,<PAD>
[21] Moyen ou max

Ensuite on peut faire la moyenne de chaque Df_i et on transforme ça en un distribution avec un softmax par exemple. Ce qui nous permet d'avoir l'influence de chaque token sur la décision de la classe.

Pooling pour la localisation

On entraîne un réseau Transformer classique mais à la place de réserver un seul token spécifique pour la classification comme le token <cls>, on classifie le plongement lexical de sortie de chaque token, ensuite on fait un pooling[22] pour déduire la classe du domaine en entrée(Figure 5). Pour la phase d'inférence on peut donc voir quel token dépasse un certain seuil de probabilité pour une certaine classe et donc on peut associer à chaque token un label et faire de la classification multi-label et en plus localiser les domaines qui appartiennent à un certaine famille.

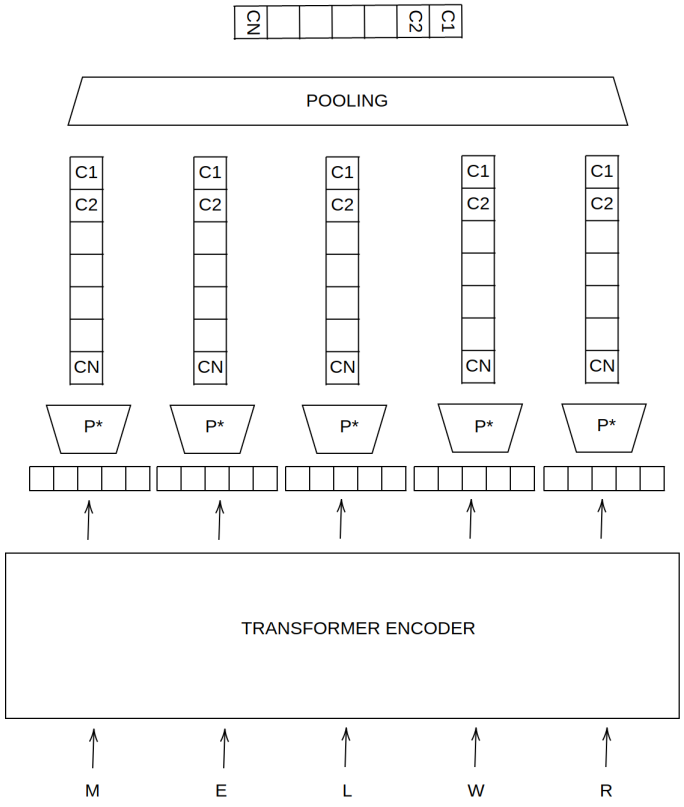


Figure 5: Réseau Transformer qui permet la localisation de la décision, avec C classes de 1 à N pour les différentes familles de protéines. P permet de projeter de la dimension du plongement lexical vers la nombre de classes.

3.6 Evaluation des tâche de classification

Pour la challenge CAFA avec les GO-annotations, des métriques spécifiques ont été développées grâce à la théorie de l'information, car les classes sont organisées selon une ontologie. Les métriques évoluent au fur est à mesure des années mais pour le CAFA de 2017 qui est le dernier où l'on dispose aussi du jeu de test. Il y a principalement deux métriques pour la qualité de la prédiction que sont le Fmax et le Smin. On peut tester tout les seuils t entre 0 et 1 avec un pas de 0.01. On peut calculer ces métriques pour les trois ontologies séparément comme dans CAFA ou prendre aussi en compte les relations entre les ontologies(notamment la

[22]Moyen ou max

relation part-of) comme dans certains articles[23].

Légende :

- f est une classe de la GO annotation
- T est l'ensemble des vraies annotations
- Pi(t) est l'ensemble des annotation prédites pour la protéine i avec un seuil de t
- m(t) est le nombre de protéines dont on a prédit au moins une classe
- n est le nombre total de protéine
- I est la fonction identité qui retourne 1 si la condition est vraie et 0 sinon
- Smin calcule la distance sémantique entre les prédictions réelles et prédites en fonction du contenu d'information(IC) de la classe.
- IC(c) Le contenu de l'information est basé sur la probabilité d'annotation de la classe c
- P(c) est l'ensemble des classes parents de la classe c
- ru(t) est l'incertitude restante moyenne
- mi(t) est la mauvaise information moyenne

$$pr_i(t) = \frac{\sum_f I\left(f \in P_i(t) \wedge f \in T_i\right)}{\sum_f I\left(f \in P_i(t)\right)} \tag{2}$$

$$rc_i(t) = \frac{\sum_f I\left(f \in P_i(t) \wedge f \in T_i\right)}{\sum_f I\left(f \in T_i\right)} \tag{3}$$

$$\mathrm{AvgPr}(t) = \frac{1}{m(t)} \cdot \sum_{i=1}^{m(t)} pr_i(t) \tag{4}$$

$$\mathrm{AvgRc}(t) = \frac{1}{n} \cdot \sum_{i=1}^{n} rc_i(t) \tag{5}$$

$$F_{\max} = \max_t \left\{ \frac{2 \cdot \mathrm{AvgPr}(t) \cdot \mathrm{AvgRc}(t)}{\mathrm{AvgPr}(t) + \mathrm{AvgRc}(t)} \right\} \tag{6}$$

$$IC(c) = -\log(\Pr(c \mid P(c)) \tag{7}$$

$$S_{\min} = \min_t \sqrt{ru(t)^2 + mi(t)^2} \tag{8}$$

$$ru(t) = \frac{1}{n} \sum_{i=1}^{n} \sum_{c \in T_i = P_i(t)} IC(c) \tag{9}$$

$$mi(t) = \frac{1}{n} \sum_{i=1}^{n} \sum_{c \in P_i(t) - T_i} IC(c) \tag{10}$$

Mais les métriques de CAFA ont aussi leurs limites, par exemple elle suppose que nous sommes sur un monde fermé, c'est-à-dire que si une fonction pour une protéine n'a pas été découverte c'est qu'elle n'existe

[23]GOLabeler (You et al., 2018b), DeepText2GO(You et al., 2018a) et DeepGOPlus(Kulmanov et al.,2019)

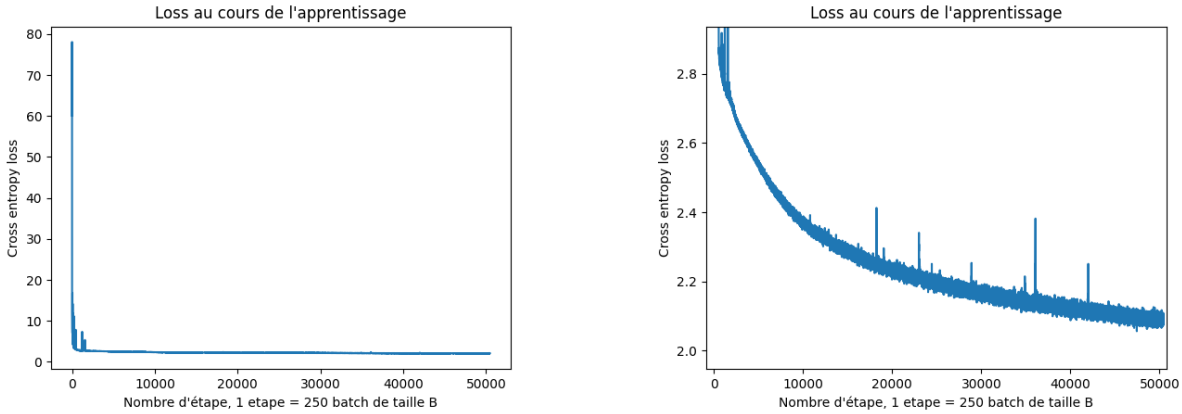
pas alors que nous devrions être sur une hypothèse de monde ouvert. Pour le prendre en compte on peut essayer de dériver des annotations ou l'expérience a prouvé que la fonction n'existait pas et ensuite avec la structure d'ontologie nous pouvons propager sur d'autre classe. Puis nous pouvons aussi utiliser les liens évolutionnaires pour propager les informations négatives, ceci n'a pas été testé dans le cadre de ce stage. La prédiction sur les GO annotations a été testée en fin de stage et donc nous avons simplifié le problème pour voir ce que cela pourrait donner. On n'a pas pris en compte la structure d'ontologie et supposé que chaque classe était indépendante. Pour cela on a utilisé une fonction de coût de cross entropie binaire. Et par la suite on a essayé de voir si le réseau arrivait avec les données d'entraînement à récupérer les relations entre les classes dans ces prédictions.

4 Résultats

Dans cette partie nous verrons les résultats des expériences décrites précédemment ainsi qu'une analyse de ceux-ci.

4.1 Phase de pré entraînement

Nous avons analysé les résultats d'un pré entraînement sur les domaines extraits de la base de données pfam. On peut remarquer dans la figure 6 que la fonction de coût descend au départ très rapidement sur les premières itérations et ensuite descend de plus en plus lentement et avec des fluctuations de plus en plus grandes. Cela signifie que notre estimation du gradient n'est plus assez précise à mesure que l'on se rapproche des paramètres optimaux qui minimisent notre fonction de coût.

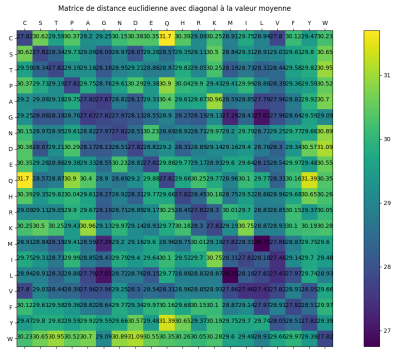


(a) Complet

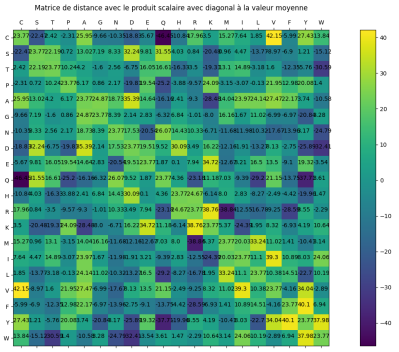
(b) Zoomé

Figure 6: Valeur de la fonction de coût au cours des itérations, lors de la phase de pré entraînement d'un Transformer

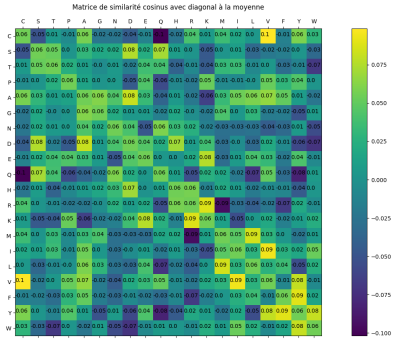
Ensuite nous pouvons voir sur la figure 7 les "distances" ou similarité pour 3 métriques différentes : cosinus, euclidienne et produit scalaire. Ces matrices sont créées à partir de la première matrice de plongement lexical fixe de notre Transformer. L'ordre des acides aminés sur la matrices a été calqué sur l'ordre d'une matrice Blosum62 qui nous sert de point de comparaison. On peut remarquer que l'on retrouve bien des similarités avec les blocs de la matrice blosum62. De plus on peut voir que l'on retrouve un partitionnement de données par propriété physico-chimique lorsque l'on projette ces plongements lexicaux en 2 dimensions avec t-SNE ou UMAP(figure 8). Ces différentes propriétés peuvent être représentées grâce à un diagramme de Venn(figure 9).



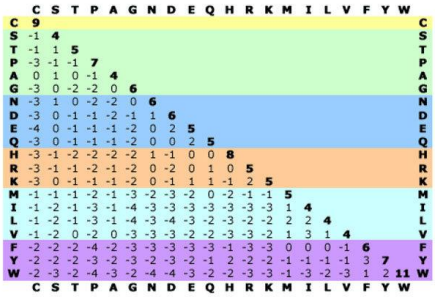
(a) Matrice distance euclidienne



(b) Matrice produit scalaire

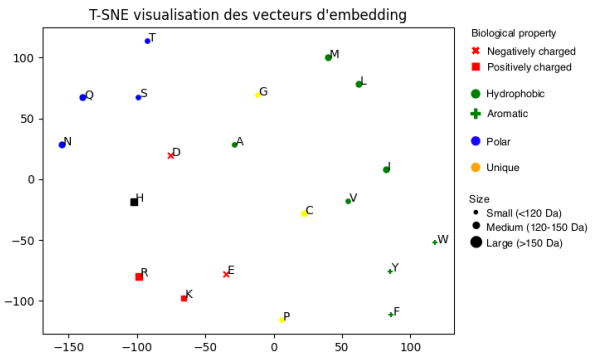


(c) Matrice similarité cosinus

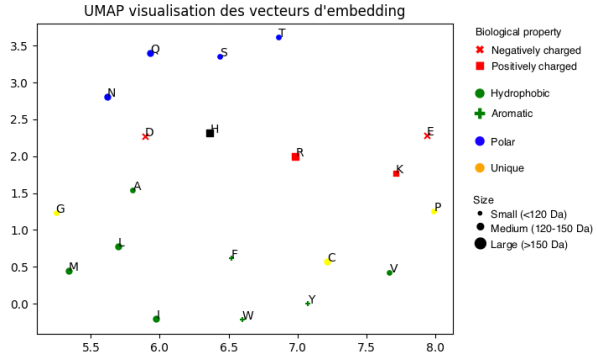


(d) Matrice Blosum62

Figure 7: Matrice de similarité ou distance sur plusieurs métriques ainsi que la matrice Blosum62 comme point de comparaison. Les matrices étant créées à partir des vecteurs de représentation fixe des acides aminés avec la matrice de plongement lexical en entrée du Transformer.



(a) projection t-SNE



(b) projection UMAP

Figure 8: Projection des vecteurs de plongement lexical d'entrée du Transformer en 2 dimensions à l'aide de deux algorithme, t-SNE et UMAP

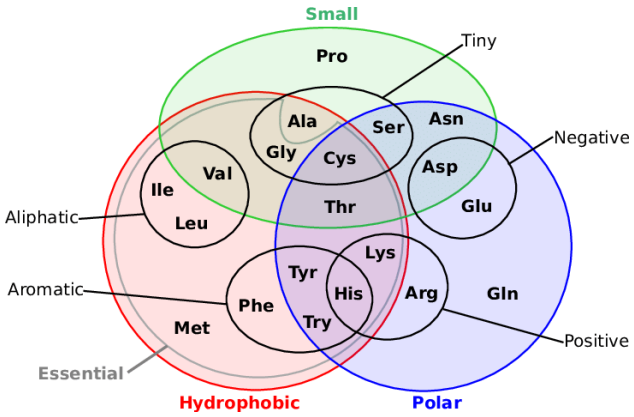
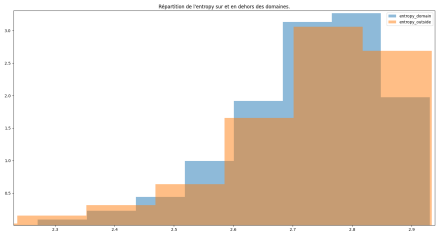
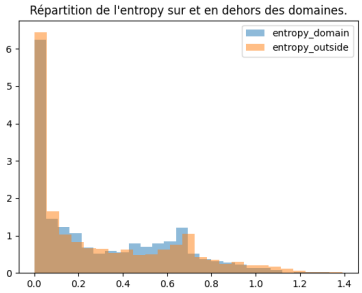


Figure 9: Diagramme de Venn montrant les différentes propriétés des acides aminés

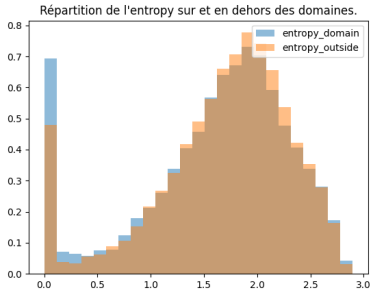
Ensuite nous avons analysé la sortie du réseau sur les tokens masqués et aussi sur les tokens non masqués avec un réseau entraîné et non entrainé (ref figure 10). On peut remarquer que sur le réseau entraîné l'entropie sur les masques possède un seul mode à environ 2.8 alors que la prédiction sur tous les tokens possède deux modes avec un très proche de zéro et l'autre aux alentours de 1.9. Cela veut dire que pour certains tokens le réseau est très sûr de lui et à donc une très faible entropie dans sa prédiction, et en moyenne le réseau est plus sûr de lui sur les tokens non masqués ce qui parait cohérent. Sur le réseau non entraîné on retrouve à peu près la même distribution d'entropie avec ou sans les masques avec un réseau très sûr de lui.



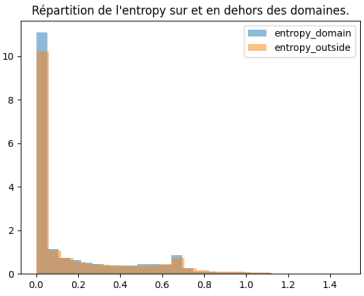
(a) Entraîné : Entropie sur les masques



(b) Non entraîné : Entropie sur les masques



(c) Entraîné : Entropie sur tous les tokens

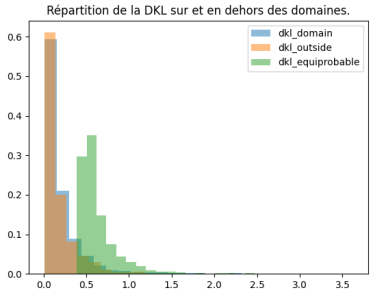


(d) Non entraîné : Entropie sur tous les tokens

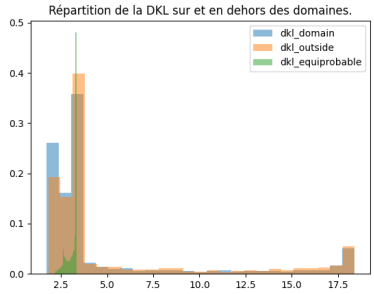
Figure 10: Diagramme représentant la distribution d'entropie avec et sans masque et sur un réseau entraîné et un non entraîné

Ensuite nous avons voulu regarder si l'on avait une différence de prédiction entre les acides aminés qui sont

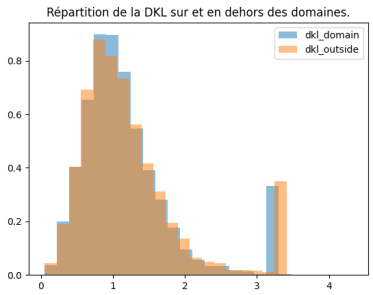
dans des domaines ou en dehors des domaines, nous avons donc calculé les divergences de Kullbach-Leibler entre la fréquence des acides aminés sur le dataset total et la prédiction de sortie du réseau(figure 11). On ne remarque presque aucune différence en moyenne pour cette divergence. Par contre lorsque l'on compare un réseau entraîné et non entraîné, on peut remarquer que la distribution des prédictions sur les masques se rapproche de la fréquence des acides aminés dans l'ensemble d'entraînement. Mais sur les prédictions hors des masques avec un réseau entraîné on peut remarquer un petit pic de divergence vers 3.2 qui n'est pas expliqué. Cela rejoint peut-être les résultats sur l'entropie où sur certains tokens le réseau est très sûr de lui même s'il s'écarte grandement de la fréquence dans le jeu de test.



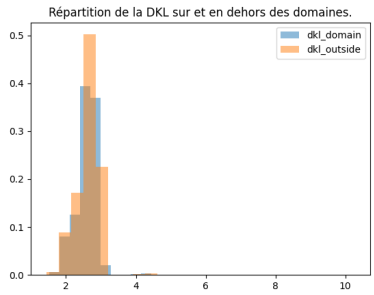
(a) Entraîné : Divergence de Kullback-Leibler sur les masques



(b) Non entraîné : Divergence de Kullback-Leibler sur les masques



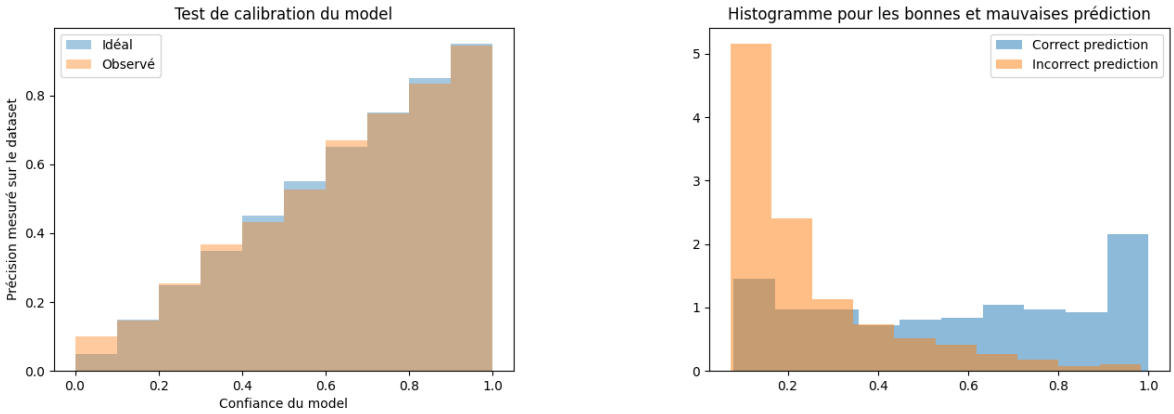
(c) Entraîné : Divergence de Kullback-Leibler sans masque



(d) Non entraîné : Divergence de Kullback-Leibler sans masque

Figure 11: Diagramme représentant la distribution des divergences de Kullbach-Leibler par rapport aux fréquences des acides aminés dans le dataset d'entraînement avec et sans masque et sur un réseau entraîné et un non entraîné, ainsi qu'avec la distribution uniforme(en vert)

On a aussi effectué des tests de calibration, comme on peut le voir sur la figure 12. On remarque que le réseau est bien calibré et n'est pas surconfiant. Lorsque la probabilité la plus haute est en dessous de 0.2, il y a plus de chances que cela soit une mauvais prédiction mais quand la probabilité est supérieure à 0.8 alors on est quasi sûr que le réseau fait une prédiction correcte.



(a) Calibration modèle

(b) Confiance selon prédiction

Figure 12: a) Diagramme représentant pour chaque tranche de confiances données par le modèle la véritable accuracy sur cette tranche. b) Probabilité maximum de sortie par le modèle en séparant les prédictions correctes et incorrectes

Nous avons aussi récupéré des métriques quantitatives à l'issue de l'entraînement du réseau (réf. tableau 1) qui sont similaires à d'autres travaux réalisés avec des Transformers.

fonction de coût	*Exponentiated cross entropy (ECE)*	Perplexité	Accuracy
2.07	7.97	11.10	0.32

Table 1: Métrique calculée sur un sous échantillon de pfam à l'issue de la phase de pré entrainement

4.2 Attention avec un Transformer non entraîné et entraîné

On a étudié un Transformer non entraîné et en cours d'apprentissage pour essayer de mieux comprendre les biais inductifs. L'analyse de l'attention avant et après entraînement nous montre bien(figure 13) qu'au départ l'attention est très répartie, elle est sur beaucoup plus de token alors qu'après entraînement elle se focalise bien sur uniquement très peu de token.

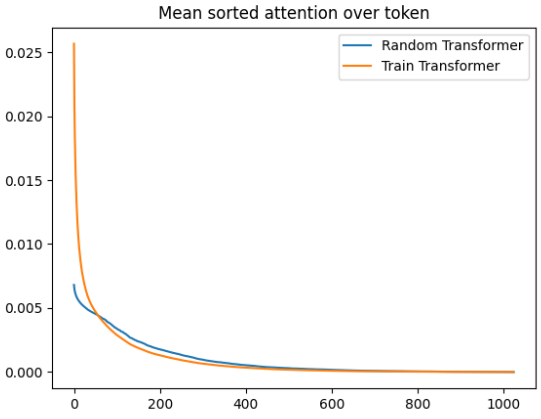


Figure 13: Répartition de l'attention en moyenne avec un réseau Transformer entraîné et non entraîné

4.3 Localisation des domaines

Les différentes architectures ont été nommées pour mieux s'y retrouver. Classique est une phase d'ajustement d'un transformer qui projette le token <cls> vers le nombre de classes. PredEach est l'architecture avec la localisation de la classification, LinAttn simplifie la dernière couche d'attention en enlevant le feedforward pour permettre une rétro-propagation de la classification sur chaque acide aminé. Pour la localisation des domaines, nos meilleurs résultats étaient avec l'architecture PredEach mais n'a rien donné de très concluant. Prédire uniquement la classe majoritaire nous donnait de meilleurs résultats comme nous pouvons le voir sur le tableau 2.

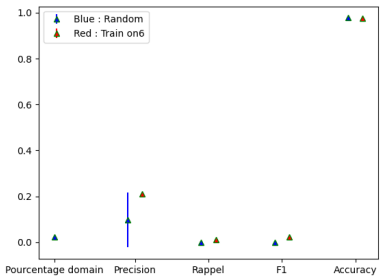
Modèle	Acc
Classique	56%
PredEach	**62%**
LinAttn	69%
Baseline (classe majoritaire)	71%

Table 2: Accuracy sur la localisation des domaines pfam.

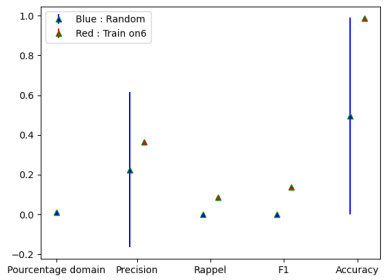
4.4 Contact 3D

Dans cette partie nous allons étudier les liens entre l'attention du réseau Transformer et les contacts 3D de la protéine. Nous avons utilisé les 3 types de contacts définis précédemment. Tout d'abord il faut savoir que même l'union des deux types de contacts représente une très faible proportion de toutes les combinaisons d'acides aminés, environ 1%.

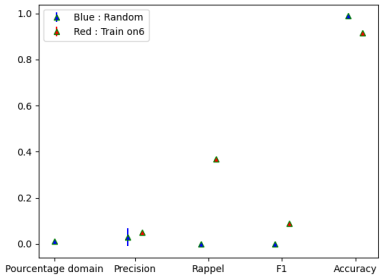
Le réseau fait plus attention aux acides aminés en contact en général, car avec tous les contacts et l'agrégation des matrices de confusion de toutes les têtes on arrive à environ 22% de précision, par contre un très faible rappel est obtenu (fig 14). Pour les contacts longue distance c'est déjà beaucoup moins le cas si l'on considère toujours la moyenne sur toutes les têtes d'attention. Mais si l'on suppose que chaque tête d'attention se spécialise pour capturer certaines propriétés, alors on peut regarder la tête qui capture le mieux cette propriété de contact 3D. En faisant ça, on arrive en sélectionnant la meilleure précision à environ 38% de précision et un rappel à 10% uniquement sur les contacts lointains. Et si on fait le max de l'attention (peu importe la tête d'attention et avec un seuil) on arrive à un rappel d'environ 40% mais la précision descend à 10%. Maintenant si l'on considère uniquement les contacts proches, alors on dispose une tête qui a une précision de quasi 100% et un rappel de 0.5%. Ce qui explique peut être pourquoi l'architecture de réseaux convolutionels (CNN) n'était pas si mauvaise avec ces connexions locales qui sont reproduites avec le Transformer, mais le réseau Transformer est bien plus polyvalent pour les relations longue distance aussi importantes.



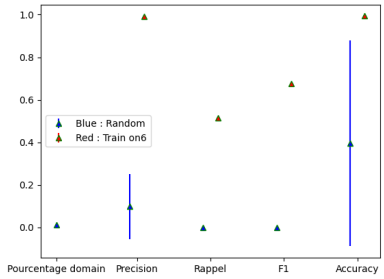
(a) Agrégation sur tous les types de contacts



(b) Meilleure précision sur uniquement les contacts lointains



(c) Maximum sur uniquement les contacts lointains



(d) Meilleure précision sur uniquement les contacts proches

Figure 14: Attention sur les contacts avec un réseau entraîné et non entraîné avec plusieurs métriques en affichant la moyenne et +/- un écart type.

4.5 Phase d'ajustement

L'analyse du réseau pré entraîné est importante mais les métriques comme l'accuracy sur la prédiction de token masqué pour la perplexité ne sont pas très bonnes pour évaluer la capacité du réseau. C'est pour cela que nous avons besoin de l'entraîner sur des tâches spécifiques pour comparer les performances. La première tâche est la classe de classification sur pfam avec une séparation de l'ensemble d'entraînement, de développement et de test avec un partitionnement comme décrit dans la partie précédente.

Pour plus de concision nous avons nommé les différentes architectures. Multispace est une architecture avec une projection de tout les plongements lexicaux de tous les acides aminés vers un espace plus grand puis un pooling maximum ou moyen entre tous les acides aminés puis une projection vers le nombre de classes. Et protCNN est le modèle présenté dans le papier *Using Deep Learning to Annotate the Protein Universe*[1].

Modèle	Accuracy
Dropout	78,8 %
Classique	72,83 %
PreadEach	69,58 %
multispace max	64,26 %
multispace mean	65,48 %
LinAttn	65,56 %
ProtCNN (Google/MIT/Cambridge University)	72,3 %
Top pick pHMM	81,9 %

Table 3: Pourcentage de prédictions correctes sur Pfam.

On remarque qu'avec l'architecture Transformer on arrive à faire mieux que le dernier réseau convolutionnel (ProtCNN) appliqué pour cette tache. Mais sans utiliser de méthodes d'ensemble nous ne parvenons pas à faire mieux que le top-pick pHMM. Il est fort probable qu'avec ces méthodes pour arriverons à faire mieux car la méthode d'ensemble de ProtCNN y arrivait mais le temps nécessaire pour ces multiples phases d'ajustements n'a pas pu être pris.

Modèle	Accuracy
Mon Transformer classique	20,6 %
ResNet	17 %
Transformer TAPE	21 %

Table 4: Pourcentage de prédictions correctes sur la tache Remote Homology Detection de TAPE

4.6 Symétrie des matrices d'attention

Pour calculer le "degré" de symétrie d'une matrice nous allons utiliser la norme 2 et calculer la distance entre la matrice et son projeté orthogonal sur l'ensemble des matrices symétriques. Avec la norme 2 on sait que les matrices symétriques et antisymétriques sont orthogonales. Et on peut donc décomposer une matrice en : $M = S + A$ avec S une matrice symétrique et A une matrice antisymétrique, $S = \frac{1}{2} * (M + M^t) et A = \frac{1}{2} * (M - M^t)$ et donc la distance est définie par la norme 2 de A($|A|_2$)

Type de réseau	Moyenne	Écart type
Réseau entraîné	1.397	1e-6
Réseau aléatoire	0.192	0.010

Table 5: Résultat de symétrie de l'attention avec un réseau entraîné et un réseau non entraîné

On peut remarquer que le réseau entraîné possède une asymétrie plus grande que le réseau initialisé aléatoirement.

4.7 Gene Ontology

Par la suite nous nous sommes intéressés à la gene ontology(GO) qui surtout sur la partie fonction moléculaire est bien plus rigoureuse au niveau de la classification que les autres ontologies. Mais il reste quelques zones d'ombre notamment plusieurs protéines avec la même séquence peuvent être présentes plusieurs fois dans la base de données avec des annotations de fonction différentes, cela est peut être dû au fait que selon l'organisme dans lequel la protéine est présente alors la fonction n'est pas la même dans certain cas. Pour savoir combien cela représentait de protéines parmi la base de données de fonctions annotées de CAFA3. Nous avons compté le nombre de ces protéines qui apparaissent plusieurs fois avec différentes annotations, et nous avons marqué quel pourcentage du nombre total de protéines annotées cela représentait (ref tableau 6).

Type	BPO	CCO	MFO
Nombre d'annotations	62342	14534	9138
Nombre de protéines avec plus d'une apparition	603	219	377
Pourcentage du total	0,96%	1,5%	4,12%

Table 6: Nombre de protéines pour chaque ontologie ainsi que le nombre de protéines doublons entre plusieurs espèces.

Nous avons aussi voulu voir si uniquement avec des probabilités indépendantes le réseau pouvait mémoriser la structure d'ontologie et les relations entre les différentes classes. D'après le tableau 7 on peut

remarquer que le réseau arrive facilement à retrouver la plupart des parents pour un seuil de probabilité faible mais lorsque l'on augmente le seuil il y a quand même une partie des classes parentes qui ne sont pas retrouvées. Cela pourrait donc être intéressant d'intégrer directement la structure d'ontologie dans la phase de prédiction du réseau ou d'essayer d'intégrer l'information dans le réseau.

Seuil	MFO	BPO	CCO
0.9	85.95%	85.21%	81.80%
0.2	97.07%	96.79%	96.73%

Table 7: Pourcentage de parent retrouvé dans les prédictions selon un seuil de probabilité sur les classes et pour les 3 ontologies

Nous avons aussi entraîné notre réseau sur la tâche qui se rapprochait le plus de ce qui nous intéresse sur le benchmark TAPE, qui est la tâche de remote homology detection et nous avons obtenu des résultats similaires aux autres travaux avec des Transformers sur cette tâche.

5 Conclusion et perspective

Nous avons vu grâce à ce stage que les outils du TAL sont adaptés à la tâche de caractérisation de protéines mais on est resté très proche de ce qui se faisait en TAL. Maintenant il est temps de prendre plus en compte les spécificités des séquence biologiques pour constituer les modèles, ainsi que de penser à l'intégration de données a priori. Sur certaines tâches nous nous approchons des résultats des pHMM qui sont beaucoup utilisés dans le domaine. Mais l'avantage du modèle Transformer est qu'il permet de prédire directement la classe au lieu d'avoir autant de pHMM que de familles différentes.

Il pourrait être intéressant de tester l'apprentissage en échantillonnant les masques de façon inversement proportionnelle à leur fréquence d'apparition dans le jeu de données, ou sinon de mettre des poids différents sur la fonction de coût selon si l'erreur se fait par rapport à un acide aminé fréquent ou peu fréquent.

De plus au cours de la thèse qui fait suite à ce stage, nous allons étudier l'impact de l'intégration de données a priori sur les performances sur certaines tâches. Quelques pistes sont envisagées, notamment on pourrait intégrer les fonctions de coût sur des tâches spécifiques directement pendant la tâche de pré entraînement ou garder la tâche de prédiction d'acide aminé pour servir de régularisation lors de la phase d'ajustement. De plus on testera aussi d'apprendre sur plusieurs tâches en même temps en prenant la sortie au niveau de plusieurs couches différentes, cela permettra au réseau d'avoir plusieurs niveaux d'abstraction et d'adapter le niveau d'abstraction à la tâche avec des poids différents selon chaque tâche.

References

[1] Maxwell L. Bileschi et al. "Using Deep Learning to Annotate the Protein Universe". In: *bioRxiv* (2019). DOI: 10.1101/626507. eprint: https://www.biorxiv.org/content/early/2019/05/04/626507.full.pdf. URL: https://www.biorxiv.org/content/early/2019/05/04/626507.

[2] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[3] Rewon Child et al. *Generating Long Sequences with Sparse Transformers*. 2019. arXiv: 1904.10509 [cs.LG].

[4] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *arXiv:1412.3555 [cs]* (Dec. 11, 2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555 (visited on 08/20/2019).

[5] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=r1xMH1BtvB.

[6] Zihang Dai et al. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". In: *arXiv:1901.02860 [cs, stat]* (Jan. 9, 2019). arXiv: 1901.02860. URL: http://arxiv.org/abs/1901.02860 (visited on 08/21/2019).

[7] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (Oct. 10, 2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 08/20/2019).

[8] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146 [cs.CL].

[9] Xiaoqi Jiao et al. *TinyBERT: Distilling BERT for Natural Language Understanding*. 2019. arXiv: 1909.10351 [cs.CL].

[10] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: 2001.04451 [cs.LG].

[11] Jun Lan et al. "Structure of the SARS-CoV-2 spike receptor-binding domain bound to the ACE2 receptor". In: *Nature* 581.7807 (2020), pp. 215–220.

[12] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2019. arXiv: 1909.11942 [cs.CL].

[13] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].

[14] Xiaodong Liu et al. *Improving Multi-Task Deep Neural Networks via Knowledge Distillation for Natural Language Understanding*. 2019. arXiv: 1904.09482 [cs.CL].

[15] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[16] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: (), p. 12.

[17] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. arXiv: 1910.10683 [cs.LG].

[18] Roshan Rao et al. *Evaluating Protein Transfer Learning with TAPE*. 2019. arXiv: 1906.08230 [cs.LG].

[19] Alexander Rives et al. "Biological Structure and Function Emerge from Scaling Unsupervised Learning to 250 Million Protein Sequences". In: *bioRxiv* (2019). DOI: 10.1101/622803. eprint: https://www.biorxiv.org/content/early/2019/04/29/622803.full.pdf. URL: https://www.biorxiv.org/content/early/2019/04/29/622803.

[20] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. arXiv: 1910.01108 [cs.CL].

[21] Nils Strodthoff et al. "UDSMProt: universal deep sequence models for protein classification". In: *Bioinformatics* 36.8 (Jan. 2020), pp. 2401–2409. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa003. eprint: https://academic.oup.com/bioinformatics/article-pdf/36/8/2401/33116463/btaa003.pdf. URL: https://doi.org/10.1093/bioinformatics/btaa003.

[22] Yu Sun et al. *ERNIE 2.0: A Continual Pre-training Framework for Language Understanding*. 2019. arXiv: 1907.12412 [cs.CL].

[23] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018. arXiv: 1804.07461 [cs.CL].

[24] Wei Wang et al. *StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding*. 2019. arXiv: 1908.04577 [cs.CL].

[25] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. arXiv: 1906.08237 [cs.CL].